

COSC 8, Fall 2007

Practice Final Exam

SAMPLE SOLUTIONS

December 7, 2007

1. (20 Points) **Streams.**

Consider a stream of the rows of Pascal's triangle, with the rows given as lists:

`[1], [1,1], [1,2,1], [1,3,3,1], [1,4,6,4,1], ...]`

To get each row, you add two copies of the previous row, but one copy is shifted by one place. Consider the entry `[1,3,3,1]`:

```
  1 3 3 1
+   1 3 3 1
-----
  1 4 6 4 1
```

Define a stream `pascalTri` that uses this approach to recursively generate itself.

Solution:

```
pascalTri = [1] : map (\lst -> zipWith (+) (0 : lst) (lst ++ [0])) pascalTri
```

2. (25 Points) Parsing.

Regular expressions are used in computer science to specify the format of things like variables, numbers, etc. A simple form of regular expression can be defined by the following BNF:

```
regExpr = letter
         | '(' regExpr regExpr ')'
         | '(' regExpr '|' regExpr ')'
         | '(' regExpr ')' '*'
```

A regular expression tree has a different constructor for each of the options in the BNF above:

```
data RegExpr = Literal Char
              | Seq RegExpr RegExpr
              | Or RegExpr RegExpr
              | Star RegExpr
  deriving Show
```

Write a parser `regExpr :: Parser RegExpr` that parses an input string into a regular expression tree. You may assume that `ParserX` has been imported, and may use any of the things that it exports:

```
module ParserX(Parser,
               (#), (-#), (#-), (#:), (#++), (!), (>->), (?), lit, letter, space,
               char, takeParse, takeWhileParse, cons, returnParse, failParse,
               takeRepeatParse, parseSpaces, parseWord, wordChoice, elimSpacesAround,
               module Data.Char)
```

Thus a call to `regExpr "(((a)*((bc)|d))*"` should return

```
Just (Star (Seq (Star (Literal 'a'))
                (Or (Seq (Literal 'b') (Literal 'c')) (Literal 'd')))), "")
```

Write your parser here:

Solution:

```
regExpr = lit '(' #- regExpr #- lit ')' #- lit '*' >-> Star
! letter >-> Literal
! lit '(' #- regExpr # regExpr #- lit ')' >-> (\(e1, e2) -> Seq e1 e2)
! lit '(' #- regExpr #- lit '|' # regExpr #- lit ')' >-> (\(e1, e2) -> Or e1 e2)
```

3. (20 Points) **Databases.**

- (a) Describe how to take the inner join of two tables. What property does the column chosen in the first table have to have? If the first table has m rows and the second table has n rows, what is the maximum number of rows that their inner join can have? \Leftarrow

Solution:

The inner join of two tables requires the column chosen in the first table to be the key of the second table. For each row in the first table we take the value in the chosen column and use it as a key to look up a row in the second table. This row must be unique if it exists. We append the two rows to make a row in the new table. (If the key is not found in the second table, no row is created.) Therefore the maximum size of the inner join is m , the size of the first table.

- (b) Describe how to take the join of two tables. If the first table has m rows and the second table has n rows, what is the maximum number of rows that their inner join can have? \Leftarrow

Solution:

A column is specified for each table. For each row in the first table we find its value v in the chosen column. Every row in the second table with same value v in its chosen column is appended to the row in the first table to create a new row in the join. If every row in the first table has the value v in its chosen column and if every row in the second table also has value v in its chosen column, then every row in the first table is paired with every row in the second table, giving mn rows in the join table.

4. (15 Points) Regions and Pictures.

(a) (10 Points) User Coordinates

Consider the following functions for converting user coordinates to window coordinates:

```
userXtoWin    :: UserWindow -> Float -> Int
userXtoWin uw x = round (intToFloat (xWin uw) * (x - userXmin uw)/userXrange uw)
```

```
userYtoWin    :: UserWindow -> Float -> Int
userYtoWin uw y = round (intToFloat (yWin uw) * (userYmax uw - y)/userYrange uw)
```

Here `xWin uw` and `yWin uw` are the width and height of the graphics window expressed in pixels, `userXmin uw` and `userYmax uw` are the minimum x value and maximum y value in the user window, and `userXrange uw` and `userYrange uw` are the width and height of the window expressed in user coordinates.

Explain why these functions correctly convert x and y values in user coordinates to the corresponding pixel values in window coordinates. In particular explain why one formula uses $(x - \text{userXmin } uw)$ and the other uses $(\text{userYmax } uw - y)$. Drawing a picture may help.

Solution:

To translate x correctly we first find how far it is from the left side of the window ($(x - \text{userXmin } uw)$) and then scale this by dividing by `userXrange uw`. This is the fraction of the window that lies to the left of x . Multiplying it by the window width (and rounding) gives the correct window x value, because $(0,0)$ is at the upper left corner of the window.

To translate y correctly we want to find how far it is from the top of the window, because the y -coordinates in the graphics window has 0 at the top and gets larger as we go down. Because the y -coordinates in the user window get smaller as we go down, we take the maximum y and subtract y . The scaling works the same way as for the x -axis.

(b) (5 Points) Tetris

In the sample solution to SA9 I kept track of the (x, y) coordinates of the center of the piece as it fell and was moved left and right. I translated the piece to the appropriate position before I drew it every time through the loop. Both `x`, `stime`, and the original piece are passed to recursive calls to `loop`. Moving the piece left or right involved updating `x`, not translating the piece.

On the other hand, when rotating the piece I simply passed (`RotateL piece`) to the recursive call to `loop` rather than keeping a count of rotations and passing that. I could have used the same strategy for changing (x, y) values. Instead of passing `x`, `stime`, and the original piece I could have passed the current time and a translated piece.

Why might this alternate design be worse than the one I used?



Solution:

The problem is that when I call `Translate` or `RotateL` I don't actually change the piece. I just create a new item with that constructor and the parameters saved. The piece translates many times (at least dozens, possibly hundreds) as it falls. Every time I draw the piece or get its vertices to check for collisions I would have to process every one of those translations. This might slow the program down enough to make the graphics jerkier.

Rotations have the same problem, but it is seldom the case that a person would rotate a piece hundreds of times as it fell! Therefore rotations are unlikely to be a problem.

5. (20 Points) **Short Answer.**

- (a) What is pruning in backtracking? Where does the name come from? Why does it speed up a backtracking program? ←

Solution: Pruning is eliminating a partial solution that cannot be expanded to a full solution. It is called pruning because it lops off an entire branch of the search tree (the partial solution and all partial solutions in its subtree). Because the number of solutions usually grows exponentially every level you go down in the search tree pruning can eliminate a lot of computation.

- (b) The List Monad is useful in non-determinism. In the second problem set you wrote a function `stepNFA` that took a set of states and generated all successor states after one step of the NFA. One way to write the code is:

```
stepNFA :: (Eq a, Eq b) => Automaton a b -> [a] -> b -> [a]
stepNFA nfa states label =
    uniques $ foldr (++) [] (map (transNFA nfa label) states)
```

where `uniques` eliminates duplicates in a list and `(transNFA nfa label)` is a function that takes a state and returns a list of states that are reachable in one step from that state following edges labeled `label`. Its principal type is `a -> [a]`, where `a` is the type of a vertex in `nfa`.

Show how to use the List Monad given below to replace everything after the “\$” in the definition of `stepNFA` by a simple monad operation. Unfold the monad command to show that it does the correct thing.

```
instance Monad [] where
  m >>= k      = concat (map k m)
  return x     = [x]
  fail x       = [ ]
```

Solution:

```
states >>= (transNFA nfa label)
```

Unfolding `>>=` gives

```
concat (map (transNFA nfa label) states)
```

which is equivalent to the expression after the “\$”.

- (c) Describe the criteria used in Norvig's spell checker to pick the word most likely to be the correct spelling of a mis-spelled word. ⇐

Solution: He first found the set of words of minimum edit distance from the word being checked. (He limited the edit distance search to 2, and if no words were that close just returned the word submitted.) From among those words he selected the one that appeared most frequently in the file `big.txt`, a compilation of a number of books downloaded from Project Gutenberg.

- (d) What would happen if the queue were replaced by a stack in BFS? Would everything reachable be found? Would the tree returned be a shortest path tree? ⇐

Solution: Instead of processing vertices in the order that they are discovered the algorithm would process the first vertex, then the first neighbor of the first vertex, then the first neighbor of that vertex, etc. This process is called depth-first search, because it goes as deep as possible before it backtracks and tries another choice. Everything reachable would be found, but the tree returned would almost never be a shortest path tree.